Recursion

CSC148, Introduction to Computer Science
Fall 2016
Diane Horton
Reasoning about correctness

• Let $P(n) =$
  “For any nested list $obj$ of depth $n$, $\text{nested\_sum}(obj)$ returns, and returns the sum of the numbers in $<obj>$.”

• We want to know that $\forall n \geq 0, P(n)$. 
Tracing from the smallest case up

• We traced the function and concluded that:
  – $P(0)$ is true.
  – $P(1)$ is true as long as $P(0)$ is true.
  – $P(2)$ is true as long as $P(0)$ and $P(1)$ are true.
    . . . and we could have continued on to show that . . .
  – $P(3)$ is true as long as $P(0)$, $P(1)$, and $P(2)$ are true.
  – And $P(4)$, $P(5)$, . . . . .

• Crucially, we did not trace the “as long as” parts.

• (We had already convinced ourselves of them.)
Reasoning more formally

• It is sufficient to show that:
  
  \( P(0) \) is true.

  \( \forall n \geq 0, \quad P(k+1) \) is true as long as \( P(0), \ldots, P(k) \) are all true.

• This is enough to allow us to conclude that:
  
  \( \forall n \geq 0, \quad P(n) \).
But don’t I have to trace everything?

• It may feel odd *not* trace the recursive call(s).
• Must you trace sort to understand this?

```python
def minimum(lst):
    """Return the minimum value in <lst>. Precondition: len(lst) >= 1 """
    copy = lst[:]
    copy.sort()
    return sorted[0]
```
• Recursion is no different.
But how does it work?!

- Of course, Python does all those function calls that we don’t have to trace.

- And these work just like any function call:
  - Push a new frame on the call stack
  - Store the parameters a local variables there
  - When the method is done, pop that frame
  - Then resume what was happening when the call was made.

- Each call has
  - its own version of the parameters and local variables,
  - in its own stack frame.

- So nothing is mixed up.
Know it, but ignore it

- You can know that this is happening.
- But you should *not* trace all those function calls.
Writing recursive functions

1. Identify the recursive structure of the data
2. Implement the base case directly
3. Pick a concrete example that is larger.
4. Write down the relevant recursive calls and what they should return
5. Assuming they do return the correct thing, figure out how to combine those results
A common pattern with Python lists

```python
i = 0
while i < len(lst):
    ... lst[i] ...
    i = i + 1
```